COBB

INSIDE VISUAL dBASE™

Tips & techniques for Visual dBASE and dBASE for Windows

October 1996 • Vol. 3 No. 10
US $8.50

Adding classes to the Form Designer

by Keith Chuvala

Visual dBASE's object-oriented language is one of its greatest strengths. The ability to create classes of objects has brought us power and flexibility previously reserved for the C++ and Smalltalk programmers of the world.

In this article, we'll show you how to create custom classes that put an elegant and easy-to-use interface on a task that otherwise might be clunky or difficult. Custom classes make life as a developer easier, and you'll get a lot of enjoyment out of sharing nifty classes with other developers.

Custom controls

A *custom control* is a special kind of custom class; it's a visual element that you can drag onto a form in the Form Designer. Visual dBASE ships with several example time-saving custom controls. The OK button, for example, is a pushbutton already set up with the OK text label and a bitmap check mark. Drop it on the form, and all you have to do is specify what should happen when the user clicks it. Simple!

Custom controls are derived from existing visual controls. In other words, custom controls have all the properties, methods, and event triggers of a "parent" control like PushButton, but you can customize or extend the custom controls in some way.

Creating a custom control is amazingly easy. The following code for a simple custom PushButton makes the computer beep when you click it (I didn't say it was useful!):

```
CLASS BEEPBUTTON OF PUSHBUTTON CUSTOM
   this.text = "&Beep!"
   this.onclick = {; ?? chr(7)}
ENDCLASS
```

In this simple example, you can see that a BeepButton has all the properties, event triggers, and methods of a PushButton. The CUSTOM keyword indicates that it will reside on the Custom page of the control palette in the Form Designer, so you can drop a BeepButton onto a form.

Custom control rules

CUSTOM works only when a control is descended from a stock visual object. Stock visual objects include Text, EntryField, ComboBox, SpinBox, ListBox, PushButton, CheckBox, RadioButton, Line, Rectangle, Browse, Editor, Image, ScrollBar, Shape, TabBox, PaintBox, OLE, and others.

A custom control can descend from any of these objects. But what if none of them fit? If that's the case, sometimes a Visual Basic control (VBX) gets the job. The DBTimer VBX you receive with Visual dBASE provides a good example. The timer itself isn't visual—it's just a clock-oriented counter. But it's handy to be able to drop a timer control on a form and to program the properties for that control from the form that uses it. (For more information about DBTimer, see "A Tale of Two Tickers" in the June 1996 issue of *Inside Visual dBASE*.)

IN THIS ISSUE

Now, what about controls we create with the dBASE language, which are often referred to as custom classes? Listing A, for example, shows a custom class that gives your applications easy access to the Windows facilities for reading and writing INI files.

I developed this class with the help of Ken "Zak" Chan while attending the Borland Developers Conference in 1995, and I've used it in every sizable dBASE application I've written since then. (Consult the comments in Listing A for details on using the class; its interface is quite simple.)

The problem with this class is that it isn't tied to the main form of my application where it belongs. After all, the main form is the form that typically reads and writes INI file information. I usually instantiate the INIFILE object in my main form's OnOpen event, but there's no evidence that the class is a part of the application when I'm working in the Form Designer. I'd much rather have an INIFILE custom control in the Control Palette, which I can drop on the form and edit from there.

Converting to a control

The solution to this location problem is surprisingly simple. An INI file doesn't have or need any of the properties of a PushButton, Rectangle, or RadioButton. The PaintBox control is tailor-made for such situations. Although originally intended to give us a generic way to hook into the Windows API, a PaintBox also makes a lovely place to hang custom class code.

To change CLASS INIFILE into a custom control, change the line

```
CLASS INIFILE
```

to read

```
CLASS INIFILE(f) of Paintbox(f) Custom
```

That's all there is to it! You can now use the Setup Custom Controls facility in the Form Designer's File menu to add the control to the Control Palette. Figure A shows the Control Palette with our new INIFILE "control" in action.

I usually like to add the line

```
this.visible = .f.
```

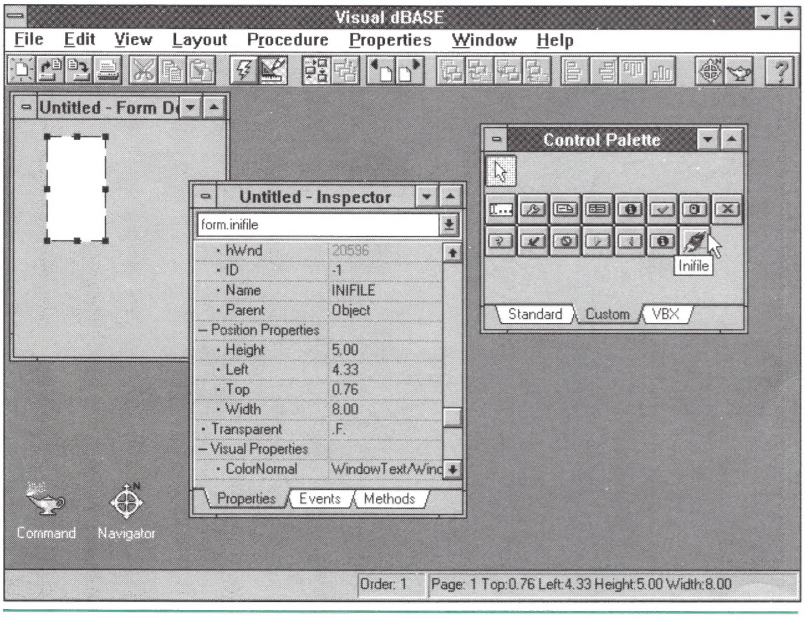
to the control's constructor code (the code that comes between the CLASS definition and the first PROCEDURE contained in the class). We don't want that PaintBox rectangle to show up while the form is running, and this line ensures that it won't.

The beauty of this trick is that you can still use the class in PRG code just as before. The fact that it's now technically a PaintBox doesn't hinder the use of the class in any way.

Control your classes

This technique doesn't make sense for every class you write, but when you'd like to have the ease of access of a custom control in the Form Designer, you can make it happen with just a few code changes. Class(es) dismissed! ❖

Figure A



Our INIFILE custom class is now a custom control. Because it's descended from a PaintBox, the icon looks the same in the Control Palette as it does in a PaintBox.

You can save time by downloading code from our ftp site at

ftp://ftp.cobb.com/ivd

Listing A: *INIFILE.CC, an INI file custom class*

```
* CLASS INI -
* wrapper for GetPrivateProfileString and
* WritePrivateProfileString
* for dBASE for Windows and Visual dBASE.
* Keith Chuvala and Kenneth Chan
* August 6, 1995
* Description: This class puts a simple interface
* on the Windows API Functions most commonly used
* to create/write and read application-
* specific .INI files.
*
* Instantiation:  set Procedure to ini
* MyINI = new INI([INI_file_name])
*
* If INI_file_name is not specified, program()
* will be used to determine path and filename
* for the .INI file, which is very
* often *not* what you want. To make an
* appropriate default .INI filename, call
* the function with SET("DIRE")+PROGRAM(1) as its
* parameter, e.g.:
* MyINI = newINI(set("dire")+"\"+program(1)).
*
* Create/Write:
* MyINI.SetValue("Section_Name","Entry","Value")
*
* If the .INI file does not exist, it will be created. The
* Section_Name section will be added to the .INI file if it
* does not already exist. Finally, an entry in the form of
* Entry=Value will be added to the [Section_Name] section.
*
* CLASS::SetValue returns the number of bytes written for the
* entry. If the return value is 0, an error occurred
* (invalid filename, inadequate rights to the directory, etc.)
*
* Read:          MyINI.GetValue("Section_Name","Entry")
*
* This Procedure returns the value requested. The return value
* (and the MyINI.Value property) will be blank if the entry was
* not found, the section does not exist, or if a problem occurred
* reading the .INI file
*
* New INI File:  MyINI.SetINIFile([INI_file_name])
*
* Notes are the same as for Instantiation, above.
*
* Properties:  INIFile: Full filename of the .INI file
* Section: Last read/written section
* Entry:   Last read/written entry
* Value:   Last read/written value
* Default: Default value for GetValue method. Normally Blank.
* Constants:     #define RETSIZE 80
* Maximum size of string returned by GetValue()

CLASS INIFILE
* Change to "CLASS INIFILE(f) of Paintbox(f) Custom"
* for custom control version!
   parameter cIniFile
* Add to ensure that control doesn't show while running!
   this.visible = .f.
   #define RETSIZE 80
   this.section = ""
   this.entry   = ""
   this.inifile = ""
   this.value   = ""
   this.default = ""
   this.name = "INIFILE"
   protect section,entry,value,inifile,default
   class::SetIniFile(cIniFile)

   if type("GetPrivateProfileString") # "FP"
      extern CLOGICAL WritePrivateProfileString(cstring, ;
       cstring,cstring,cstring) KERNEL
      extern CINT GetPrivateProfileString(cstring,;
       cstring,cstring,cptr,CINT,cstring) KERNEL
   endif

   Procedure GetValue
      parameter cSection,cEntry
      this.value = space(RETSIZE)
      if type("cSection") == "C" .and. ;
          type("cEntry") == "C"
         this.Section = cSection
         this.Entry   = cEntry
         getprivateprofilestring(this.Section,;
           this.Entry,this.Default,this.Value, ;
           RETSIZE,this.inifile)
      endif
   return trim(this.value)

   Procedure SetValue
      parameter cSection,cEntry,cNewValue
      local nBytes
      if type("cSection") == "C" .and. ;
           type("cEntry") == "C"  .and. ;
           type("cNewValue") == "C"
         this.section = cSection
         this.entry = cEntry
         this.value = ltrim(trim(cNewValue))
         nBytes=writeprivateprofilestring(this.Section,;
         this.Entry,this.Value,this.inifile)
      else
         nBytes = 0
      endif
   return nBytes

   Procedure SetIniFile
      parameter cIniFile
      local p
      if type("cIniFile") <> "C"
         this.inifile=set("directory")+"\"+program(1)
      else
         this.inifile = cIniFile
      endif
      this.inifile = trim(ltrim(this.inifile))
      if (at("\",this.inifile) = 0) .and. ;
          (at(":",this.inifile) = 0)
         this.inifile = home()+this.inifile
      endif
      if right(upper(this.inifile),4) <> ".INI"
         p = at(".",this.inifile)
         if p > 0
            this.inifile = stuff(this.inifile,p,4,".INI")
         else
            this.inifile = this.inifile + ".INI"
         endif
      endif
ENDCLASS
```

dBASE to HTML made easy

by Keith Chuvala

Web here, Web there, Web everywhere! These days, it seems as though all of my contract programming clients are interested in publishing information on the World Wide Web. In the July 1996 article "dWorld Wide Web: A First Look," I recounted my preliminary exploits with Web server software that calls a Visual dBASE application for interactive use.

I've also noticed increased interest lately in having static information, such as tables of data produced daily or weekly from a DBF table or query, to appear as a table. Creating a table with Hypertext Markup Language (HTML), the language of the Web, is straightforward, but it's also time-consuming and tedious.

Being a big fan and avid user of Visual dBASE's object-oriented language, I created a custom class that simplifies and streamlines the process of creating Web pages. Listing A shows WEBPAGE.PRG, a fairly sizable class that handles most of the details of HTML coding.

Pages made easy

Once you've entered the code presented in WEBPAGE.PRG, you can test drive it from the dBASE Command window before trying it in an application. First, add WebPage to dBASE's procedure list with a command in the form

```
SET PROCEDURE TO WEBPAGE ADDITIVE
```

The ADDITIVE parameter ensures that any other procedure files you might have open will remain open. I'm partial to the NEW command syntax, so I'll use that in this article. You can also use DEFINE syntax to create the WebPage class, if you prefer. Let's create a simple HTML document using the WebPage class:

```
w = new webpage()
w.setcomment("A page created with the WebPage
➥ class")
w.settitle("A Simple Sample Web Page")
w.centeron()
w.heading(1,"Created with Visual dBASE!")
w.separator()
w.addtext("This is a very simple page, with no
➥ dBASE data added to it")
w.htmloutput("demo.htm")
```

Let's look at each line and its specific function. The first line

```
w = new webpage()
```

creates a variable called w that contains all the properties and methods of the WebPage class. (Remember that a *method* is a procedure or function that belongs to a particular class.)

It's always nice to document your HTML code with comments. It's not required, but it's never a bad idea. The line

```
w.setcomment("A page created with the
   ➥WebPage class")
```

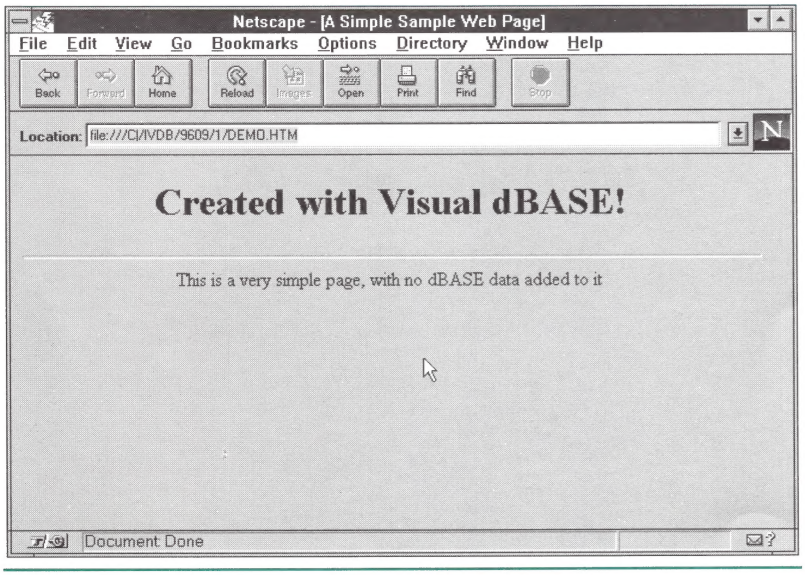
will add an HTML comment; the text won't appear on the page, but instead will appear as internal documentation.

The next line

```
w.settitle("A Simple Sample Web Page")
```

sets the title that most browsers display in the title bar. Figure A shows a sample page as viewed in Netscape. Notice the text in the title bar of the window.

Figure A



We created this page from the HTML code in Listing B.

Continued on page 8

Listing A: *WEBPAGE.PRG*

```
* WEBPAGE.PRG
* Object wrapper for producing HTML pages from Visual dBASE
*
*   Methods of the class and the arguments they take:
*
* clearpage()   && Starts a fresh page, clears all content!
* settitle("Web Page Title goes here")
* setbody("Extra parameters for the body go here")
* setbgcolor("Specify a background color in FFFFFF format or
* leave blank for color picker")
* setlinkcolor("Specify a link color")
* setvlinkcolor("Specify a visited link color")
* settextcolor("Specify a text color")
* setbackground("Specify a background graphic file")
* setcomment("Web page comment goes here")
* setaddress("Address text goes here")
* addtext("Add this line to the body of the page")
* separator("options") && inserts an <HR> horizontal rule separator
* linebreak() && Inserts a <BR> line break
* paragraph("options") && Insert a <P> paragraph break
* italics("This text will be italicized")
* bold("This text will be bold")
* emphasis("This text will be emphasized")
* strong("This text will be strong")
* heading(1-6,"This text will be set to the indicated heading
* size (1-6)")
* addlink("Specify URL here","This text will be displayed")
* centeron()  && Turns on centering
* centeroff() && Turns off centering
* addulist({"The array","specified Here","will be displayed",
* "in an unordered list!"})
* addolist({"The array","specified Here","will be displayed",
* "in an ordered list!"})
* addimage("image.filename goes here","Options (border, etc.)
* go here")
* addtable("Caption for table",{"Array of fields","or blank
* for all"},"Table Options")
* htmloutput("Filename to send Web page source to")
*
* Additional PUBLIC procedures
*
* Color2HTML("Color in 255,255,255 format, like that returned
* from getcolor()")
* italics("Returns this as an italiziced string")
* bold("Returns this as an bolded string")
* emphasis("Returns this as an emphasized string")
* strong("Returns this as an strong-ized string")
* ulist({"Returns this array","in a single string for",
* "and unordered list"})
* olist({"Returns this array","in a single string for",
* "and ordered list"})
*
#define CRLF  chr(10)+chr(13)

class webpage
   this.head       = ""
   this.title      = ""
   this.body       = ""
   this.bgcolor    = ""
   this.background = ""
   this.textcolor  = ""
   this.linkcolor  = ""
   this.vlinkcolor = ""
   this.address    = ""
   this.view       = ""
   this.alias      = ""
   this.comment    = ""
   this.aBody      = new array()

protect head,title,bgcolor,fgcolor,linkcolor,textcolor,vlinkcolor,;
        address,view,alias,comment,background,aBody

procedure clearpage
   this.head       = ""
   this.title      = ""
   this.body       = ""
   this.bgcolor    = ""
   this.background = ""
   this.textcolor  = ""
   this.linkcolor  = ""
   this.vlinkcolor = ""
   this.address    = ""
   this.view       = ""
   this.alias      = ""
   this.comment    = ""
   this.aBody.release()
   this.aBody      = new array()
return

procedure settitle
   parameter cTitle
   if type("cTitle")  == "C"
      this.Title = cTitle
   endif
return

procedure setbody
   parameter cBody
   if type("cBody") == "C"
      this.Body = cBody
   endif
return

procedure setbgcolor
   parameter cColor
   if type("cColor") <> "C"
      cColor = getcolor("Select BACKGROUND Color for the Web Page")
   endif
   this.bgcolor = Color2HTML(cColor)
return

procedure setlinkcolor
   parameter cColor
   if type("cColor") <> "C"
      cColor = getcolor("Select LINK Color for the Web Page")
   endif
   this.linkcolor = Color2HTML(cColor)
return

procedure setvlinkcolor
   parameter cColor
   if type("cColor") <> "C"
      cColor = getcolor("Select VISITED LINK Color for the Web Page")
   endif
   this.vlinkcolor = Color2HTML(cColor)
return

procedure settextcolor
   parameter cColor
   if type("cColor") <> "C"
      cColor = getcolor("Select TEXT Color for the Web Page")
   endif
   this.textcolor = Color2HTML(cColor)
return
```

```
procedure setbackground
   parameter cBackground
   if type("cBackground") <> "C"
      cBackground = getfile("*.GIF;*.JPG")
   endif
   this.bgcolor = cBackground
return

procedure setcomment
   parameter cLine
   if type("cLine") == "C"
      this.comment = cLine
   endif
return

procedure setaddress
   parameter cLine
   if type("cLine") == "C"
      this.address = cLine
   endif
return

procedure addtext
   parameter cLine
   if type("cLine") == "C"
      this.aBody.add(cLine)
   endif
return

procedure separator
   parameter cExtra
   this.addtext("<HR"+iif(type("cExtra")=="C",cExtra,"")+">")
return

procedure linebreak
   this.addtext("<BR>")
return

procedure paragraph
   parameter cExtra
   this.addtext("<P"+iif(type("cExtra")=="C",cExtra,"")+">")
return

procedure italics
   parameter cLine
   if type("cLine") == "C"
      this.addtext("<I>"+cLine+"</I>")
   endif
return

procedure bold
   parameter cLine
   if type("cLine") == "C"
      this.addtext("<B>"+cLine+"</B>")
   endif
return

procedure emphasis
   parameter cLine
   if type("cLine") == "C"
      this.addtext("<EM>"+cLine+"</EM>")
   endif
return

procedure strong
   parameter cLine
   if type("cLine") == "C"
      this.addtext("<STRONG>"+cLine+"</STRONG>")
   endif
return

procedure heading
   parameter nHead,cLine
   if (type("nHead") == "N") .and. (type("cLine") == "C")
      cHead = ltrim(str(nHead))
      this.addtext("<H"+cHead+">"+cLine+"</H"+cHead+">")
   endif
return

procedure addlink
   parameter cURL,cText
   if type("cURL") == "C" .and. type("cText") == "C"
      this.addtext('<A HREF="'+ltrim(cURL)+'">'+cText+"</A>")
   endif
return

procedure centeron
   this.addtext("<CENTER>")
return

procedure centeroff
   this.addtext("</CENTER>")
return

procedure addulist
   parameter aList
   local i
   if type("aList") == "A"
      this.addtext(ulist(aList))
   endif
return

procedure addolist
   parameter aList
   local i
   if type("aList") == "A"
      this.addtext(olist(aList))
   endif
return

procedure addimage
   parameter cImg, cOptions
   if type("cImg") == "C"
      if isblank(cImg)
         cImg = getfile("*.gif;*.jpg","Select an Image")
      endif
      if type("cOptions") <> "C"
         cOptions=""
      endif
      this.addtext('<IMG SRC="'+cImg+'"' + cOptions+">")
   endif
return

procedure addtable
   parameter cCaption,aWhichFields,cTableOpts
   local i,aFields
   private uData, cField, cLine

   cCaption    = iif(type("cCaption")   <> "C","",cCaption)
   cTableOpts  = iif(type("cTableOpts") <> "C","",cTableOpts)
   if type("aWhichFields") == "A"
      aFields = aWhichFields
   else
      aFields = new array()
      for i = 1 to fldcount()
         aFields.add(field(i))
      next
   endif
```

```
   this.addtext("<TABLE BORDER=1" + cTableOpts + ">")
   if .not. isblank(cCaption)
      this.addtext("<CAPTION>"+cCaption+"</CAPTION>")
   endif
   do while .not. eof()
      this.addtext("<tr>")
      for i = 1 to aFields.Size
         cLine = "<td>"
         cField = aFields[i]
         uData = &cField.
         cType = type("uData")
         do case
            case cType $ "CM"
               cLine = cLine + trim(uData)
            case cType $ "FN"
               cLine = cLine + ltrim(str(uData))
            case cType = "D"
               cLine = cLine + dtoc(uData)
            case cType = "L"
               cLine = cLine + iif(uData,"Y","N")
         endcase
         this.addtext(cLine + "</td>")
      next
      this.addtext("</tr>")
      skip
   enddo
   this.addtext("</TABLE>")
return

procedure htmloutput
   parameter cFile
   local i
   if type("cFile") <> "C"
      cFile = putfile("Save HTML Source as...","*.HTM")
   endif
   if .not. isblank(cFile)
      set alternate to (cFile)
      set alternate on
   endif
   ?? "<HTML>"
   if .not. isblank(this.comment)
      ? "<!* "+this.comment+"->"
   endif
   ? "<HEAD><TITLE>" + this.title + "</TITLE></HEAD>"
   ? "<BODY " + iif(.not. isblank(this.bgcolor),;
                  ' BGCOLOR="'+ltrim(this.bgcolor)+'"',"") + ;
                iif(.not. isblank(this.linkcolor),;
                   ' LINK="'+ltrim(this.linkcolor)+'"',"") + ;
                iif(.not. isblank(this.vlinkcolor),;
                   ' VLINK="'+ltrim(this.vlinkcolor)+'"',"") + ;
                iif(.not. isblank(this.textcolor),;
                   ' TEXT="'+ltrim(this.textcolor)+'"',"") + ;
                ">"
   for i = 1 to this.aBody.size
      ? this.aBody[i] + " "
   next
   if .not. isblank(this.address)
      ? "<HR>"
      ? this.address
   endif
   ? "</BODY></HTML>"
   ? " "
   if .not. isblank(cFile)
      set alternate to
      set alternate off
   endif
return
endclass

procedure italics
   parameter cLine
   if type("cLine") == "C"
      return "<I>"+cLine+"</I>"
   endif
return ""

procedure bold
   parameter cLine
   if type("cLine") == "C"
      return "<B>"+cLine+"</B>"
   endif
return ""

procedure emphasis
   parameter cLine
   if type("cLine") == "C"
      return "<EM>"+cLine+"</EM>"
   endif
return ""

procedure strong
   parameter cLine
   if type("cLine") == "C"
      return "<STRONG>"+cLine+"</STRONG>"
   endif
return ""

procedure ulist
   parameter aList
   local i,cString
   cString = "<UL>"
   if type("aList") == "A"
      for i = 1 to aList.Size
         cString = cString + CRLF+"<LI>"+aList[i]
      next
       cString = cString + CRLF+"</UL>"+CRLF
   endif
return cString

procedure olist
   parameter aList
   local i,cString
   cString = "<OL>"
   if type("aList") == "A"
      for i = 1 to aList.Size
         cString = cString + CRLF+"<LI>"+aList[i]
      next
       cString = cString + CRLF+"</OL>"+CRLF
   endif
return cString

procedure Color2HTML
   parameter cColor
   cRetVal=""
   if type("cColor") <> "C"
      return ""
   endif
   do while .not. isblank(cColor)
      p = at(",",cColor)
      if p > 0
         cPart = left(cColor,p-1)
         cColor = substr(cColor,p+1)
      else
         cPart = cColor
         cColor = ""
      endif
      cRetVal = cRetVal + itoh(val(cPart))
   enddo
return cRetVal
```

Continued from page 4

To make the page visually appealing (okay, that might be a bit of a stretch), the next line, w.centeron( ), instructs the Web page to center everything from this point forward (or until you make a centeroff( ) call).

The first thing we'll center is a heading. HTML supports 6 types of headings, numbered 1 through 6, where 1 is the largest and boldest style, and 6 the smallest. The line

```
w.heading(1,"Created with Visual dBASE!")
```

adds a level 1 heading to the page. Note that we use two arguments in the function. The first is a number between 1 and 6 (inclusive), and the second is the text of the heading. Visual dBASE automatically turns off the heading style after you place the text on the page, so there's no need to select a normal text style after this line.

Next, the line w.separator( ) inserts an <HR> tag into the HTML source. The <HR>, or horizontal rule, simply displays a separator across the page. This tag can take optional parameters, and if you're familiar with those parameters, you may pass them in a string to the separator( ) method. For instance, you might make a call in the form w.separator("width=50%").

So far, almost everything we've added has been special text or a formatting feature. The next line

```
w.addtext("This is a very simple page, with
➥ no dBASE data added to it")
```

adds some plain old text to the page with no extra bells and whistles. The addtext( ) method is the workhorse function that builds the Web page line by line. In fact, almost all of the other methods in the WebPage class use addtext( ) to insert data into the HTML document.

At this point, we're done with our sample page, so we stream the data out to an HTML file with the line

```
w.HTMLOutput("demo.htm")
```

If you don't specify a filename, the method will prompt you for one. If you don't make a selection, the method will stream the output to the Command window's Results pane. The result of this method call appears in **Listing B**. If you "speak" HTML, this listing will be immediately familiar, since we haven't done anything fancy (yet!).

Listing B: *A Simple HTML file built with the WebPage class*

```
 <HTML>
<!* A page created with the WEBPAGE class->
<HEAD><TITLE>A Simple Sample Web Page</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>Created with Visual dBASE!</H1>
<HR>
This is a very simple page, with no dBASE data
➥ added to it
</BODY></HTML>
```

Loads of methods

The WebPage class contains more than two dozen methods. Many are very simple. For instance, separator( ) simply adds a separator; centeron( ) and centeroff( ) turn centering on and off, respectively, as we've already seen. Other methods are a bit more involved, and the comments at the top of the listing explain what arguments each function requires.

Let's look at a couple of the more involved methods. The addulist( ) and addolist( ) methods create formatted lists. Specifically, addulist( ) creates an unordered, or bulleted, list while addolist( ) creates an ordered, or numbered, list. Each of these functions expects an array of values in the parentheses. While this usage might sound convoluted at first, remember that dBASE now supports

literal arrays, so we can add an ordered list, for example, with the line

```
w.addolist({"Get out of bed","Shower","Eat
    breakfast","Cook breakfast","Wish you'd
    done the previous two in the correct
    order"})
```

Adding images and links

You can easily add graphics files to the Web page with the AddImage( ) method. Pass the image's filename as a string, as in the command

```
w.addimage("logo.jpg")
```

Remember that the Web is a cross-platform phenomenon; it's important to stick to graphics formats supported by any computer that will access your web page. For example, GIF and JPG files have become the overwhelming favorites of the Web because you can view them on almost any computer that has graphics capability.

The World Wide Web is a web because of hypertext links, and the WebPage class has a method for adding those too. Invoke the addlink( ) method, passing two strings: the first string is the URL (Uniform Resource Locator, or Web page address) to jump to when the link is selected, and the second is the text to display for the link. For example, to create a link for Borland International's home page, we could add code like this to our program:

```
w.addtext("Looking for tech support?  Go visit")
w.addlink("http://www.borland.com",Borland On-
➥  Line")
w.addtext(" or call their tech support phone
➥  number!")
```

Notice how I've placed the AddLink( ) call between a pair of calls to AddText( ). Rarely does a link stand alone. With the WebPage class, you simply call each method in the order that the data needs to flow onto the Web page.

Color: shades of trouble

Methods that add color also deserve special mention. You express HTML color codes as three pairs of hexadecimal values to specify red, green, or blue intensity. Hence, FF0000 is solid red, 00FF00 is solid green, and 0000FF is solid blue. All other colors mix and match these values. For example, yellow is FFFF80, gray is C0C0C0, and so on.

Easy to remember? Not at all! Add to the confusion the fact that dBASE's color picker, called with the GetColor( ) function, returns the color as a string of comma-separated decimal numbers instead of concatenated hexadecimal numbers. (For more information about using colors in Visual dBASE, see "Working with RGB Colors" in the August 1996 issue.)

The function Color2HTML( ) is included in the WEBPAGE.PRG listing. This function takes a color set from the GetColor( ) function and converts it to an HTML-friendly color string. This function isn't encapsulated in the WebPage class; however, you can use it freely as long as WEBPAGE.PRG is in dBASE's procedure list.

Adding table data to the page

Perhaps the most time-saving method in the WebPage class is AddTable( ). This method takes three parameters: a table caption or heading, an array that lists the fields you'd like to present, and any extra table options desired by those who are HTML-savvy. The following code lets you add some customer data from the Visual dBASE sample files to your Web page:

```
use \vdb\samples\customer
w.addtable("CustomerList",
➥  {"customer_n","name","ytd_sales"},"")
w.htmloutput("cust.htm")
```

In this example, I didn't set any filters, indexes, or other conditions. As a result, the table will include all records, and dBASE will add the records to the HTML file in natural—unsorted—order. Turn to page 10, and you'll find the HTML output for this code, which appears in **Listing C**, and the resulting Netscape screen, shown in **Figure B**. To include all fields, pass a null string or any non-array argument as the second item in the parentheses in the form *w.addtable("Customer List","","")*.

Pulling out all the stops

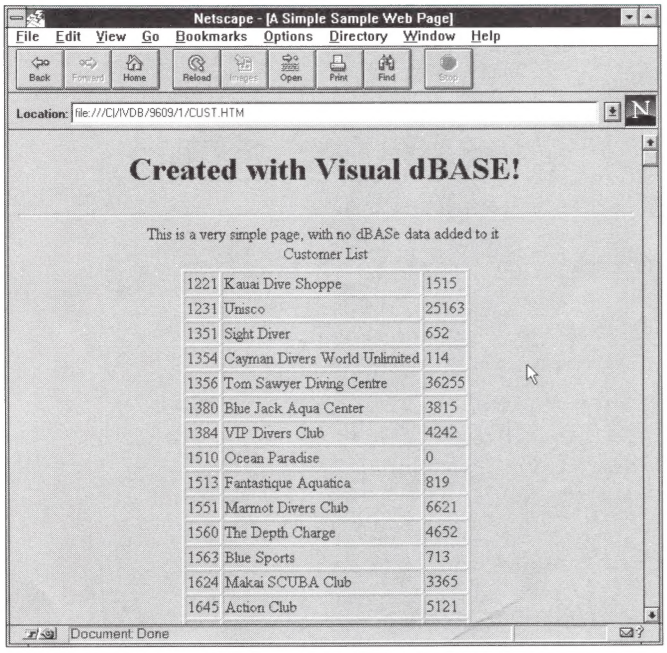
Enter the code below to generate a whiz-bang Web page that uses the same CUSTOMER.DBF table. You can type this code into a PRG file or try it right in the

Command window. Pay close attention to the use of the custom methods coded into the WebPage class and the effect these methods have on the final output.

```
clear all
set procedure to webpage additive
use customer order name
set filter to ytd_sales > 1000
go top
```

```
w = new webpage()
w.settitle("The Dive Shop: Our Best Customers!")
w.setbgcolor("ffffff")
w.centeron()
w.addimage("diveshop.gif")
w.heading(1,"The Dive Shop")
w.heading(2,"Customers with more than $1,000 YTD
➥Sales")
w.separator()
w.paragraph()
w.addtable("Customer Number, Name, and
➥Sales",{"customer_n","name","ytd_sales"},"width=80%")
w.paragraph()
w.addtext("Send all questions to ")
w.addlink("mailto:webmaster@diveshop.com",
➥"Wally Webworker")
w.addtext(" here at the Dive Shop!")
w.htmloutput("diveshop.htm")
```

Figure B



Now we have data in this Web browser's view of the code in Listing C.

We added an image to the top of the page by using the AddImage( ) method, and an E-mail hypertext link to the bottom of the page by using AddLink( ). See **Listing D** for the generated HTML code, and **Figure C** for the visual result.

WWWrapping up

In a future issue, we'll take a look at WebTools, a set of utilities now available with the Visual dBASE Professional package from Borland. WebTools expands some of the ideas presented in this article and adds a visual design surface for creating Web pages. Until you have that package in hand, experiment with the WebPage class presented here. If you come up with new ideas or would like to see our class extended, let us know. ❖

Listing C: *Simple page with table data added (edited for length!)*

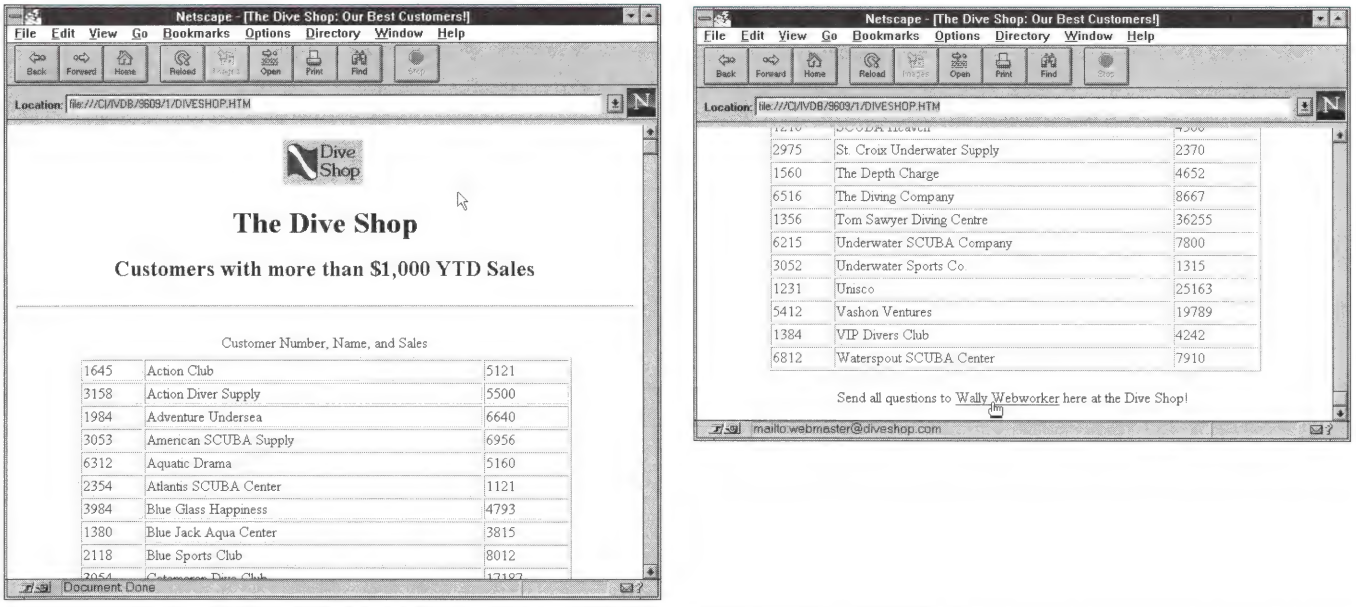
```
<HTML>
<!* A page created with the WEBPAGE class->
<HEAD><TITLE>A Simple Sample Web Page</TITLE></HEAD>
<BODY >
<CENTER>
<H1>Created with Visual dBASE!</H1>
<HR>
This is a very simple page, with no dBASE data added to it
<TABLE BORDER=1>
<CAPTION>Customer List</CAPTION>
<tr>
<TABLE BORDER=1>
<CAPTION>Customer List</CAPTION>
<tr>
<TABLE BORDER=1>
<CAPTION>Customer List</CAPTION>

</BODY></HTML>
```

```
<tr>
<td>1221</td>
<td>Kauai Dive Shoppe</td>
<td>1515</td>
</tr>
<tr>
<td>1231</td>
<td>Unisco</td>
<td>25163</td>
</tr>
<tr>
<td>1351</td>
<td>Sight Diver</td>
<td>652</td>
</tr>

</TABLE>
```

Figure C



We used the AddLink( ) method to create this E-mail link at the bottom of the Web Page.

Listing D: *A filtered, ordered, and colorized Web page*

```
<HTML>
<HEAD><TITLE>The Dive Shop: Our Best Customers!</TITLE></HEAD>
<BODY  BGCOLOR="ffffff">
<CENTER>
<IMG SRC="diveshop.gif">
<H1>The Dive Shop</H1>
<H2>Customers with more than $1,000 YTD Sales</H2>
<HR>
<P>
<TABLE BORDER=1 width=80%>
<CAPTION>Customer Number, Name, and Sales</CAPTION>
<tr>
<td>1645</td>
<td>Action Club</td>
<td>5121</td>
</tr>
<tr>
<td>3158</td>
<td>Action Diver Supply</td>
<td>5500</td>
</tr>
</TABLE>
<P>
Send all questions to
<A HREF="mailto:webmaster@diveshop.com">Wally Webworker</A>
 here at the Dive Shop!
</BODY></HTML>
```

Easy and unique temporary files

I use temporary files from time to time, especially for custom reporting purposes. In a LAN/multiuser environment, it's important to prevent two users from accessing the same temporary file at the same time—otherwise, chaos and incorrect data can result.

Visual dBASE's Funique( ) function provides a nice solution. Funique( ) returns a filename that's guaranteed not to exist in the current directory at the time you call the function. The function takes a single string argument that specifies the appearance of the string. For example, I entered the lines marked with two greater-than symbols (>>), shown below, in the Command window. dBASE displayed the unmarked lines in the Results pane:

```
>> ? funique( )
47039252

>>? funique("TEMP????.DBF")
TEMP7990.DBF

>> ? funique("????????.TMP")
79861619.TMP
```

Remember to clean up after yourself. You should delete temporary files from the disk as soon as you no longer need them. dBASE's DELETE FILE command works well for that kind of cleanup.

So, the next time you need a temporary file, check out Funique( ). Take the worry out of generating unique filenames!

Undo for your data

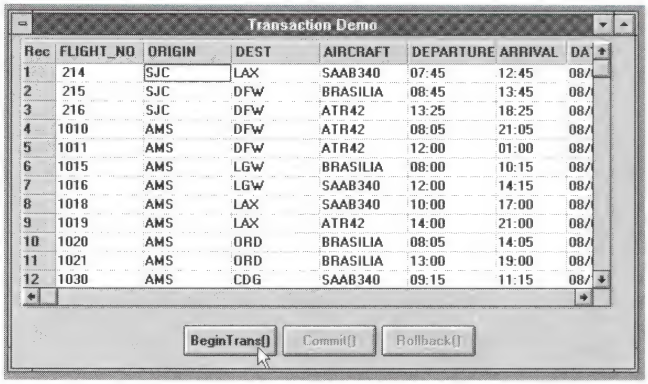
Transaction processing! To some dBASE programmers, it can sound ominous, important, hard to understand, and a bit mysterious—all at once. And it certainly is important when critical data is at stake.

But you needn't be a data-modeling expert to start using and benefiting from Visual dBASE's transaction-processing capabilities. A *transaction* is a single change or set of related changes you make to a database. The changes can take place in one table or multiple tables, but in Visual dBASE, each transaction is active for the current session. Transaction processing involves three stages:

- Marking the state of the database.
- Making change(s) to the table(s).
- Accepting or rejecting the changes and then closing the transaction.

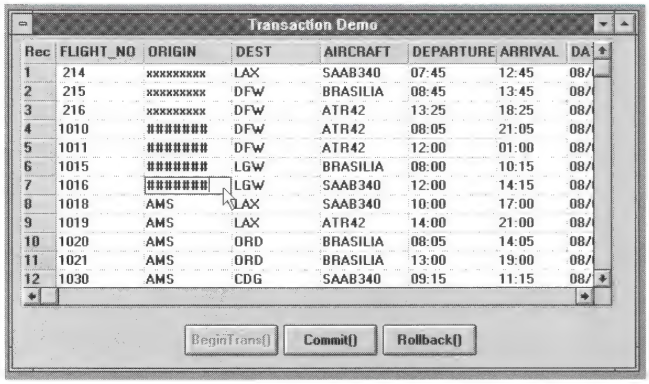
Because multiple sessions can be active simultaneously—particularly when multiple forms are open—it's important to remember that each transaction is active for its session only, and changes made in other sessions aren't affected. This behavior also means that you can't nest transactions; if you attempt to begin a second transaction before closing a prior one, Visual dBASE will display a dialog box that politely declines to do so.

Figure A



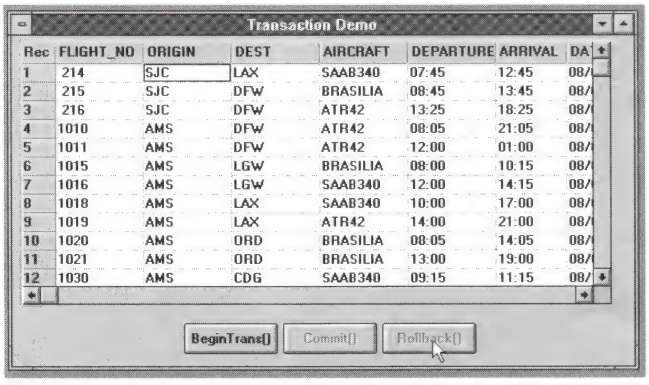
The data's in good shape, so we'll press the BeginTrans( ) button before making changes.

Figure B



Changes occur while the transaction is active. Ugly stuff!

Figure C



After you press the Rollback( ) button, all data returns to its original state.

The magic trio

You manage transactions in Visual dBASE using only three functions. BeginTrans( ) marks the beginning of a transaction, returning .T. if it's successful or .F. if the transaction can't be initiated. It's a good idea to verify that a transaction started properly by testing this return value.

Once you've made changes, you accept them and close the transaction by executing the Commit( ) function. Like BeginTrans( ), Commit( ) returns a .T. value on success.

If you need to reject the changes—in other words, undo them—execute the RollBack( ) function. Rollback( ), too, will return .T. if all goes well. This function lets you put an "undo" function in your programs.

Typically I'll run BeginTrans( ) when a form opens for data entry, and if the user pushes the Abandon Changes button, the form executes a RollBack( ) to undo any changes. If the user selects the OK or Save Changes button, the program calls Commit( ) instead.

Transaction demos

You can demonstrate the power and simplicity of transactions in the Command window. I'll use the FLIGHTS.DBF table from the Visual dBASE sample programs directory. Enter the following commands in the Command window:

```
* Entered in the Command window:
use flights
BeginTrans()
? flights->aircraft
* Original value is 'SAAB340'
replace flights->aircraft with "Yugo"
* Yes, I know this is blasphemy!
? flights->aircraft && Displays "Yugo"
Rollback()
? flights->aircraft && Displays "SAAB340" again!

* Displayed in the Results pane:
SAAB340
Yugo
SAAB340
```

Listing A: *TTEST.WFM*

```
** END HEADER * do not remove this line*
* Generated on 06/21/96
*
parameter bModal
local f
f = new TTESTFORM()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS TTESTFORM OF FORM
   this.OnOpen = CLASS::FORM_ONOPEN
   this.View = "FLIGHTS.DBF"
   this.Text = "Transaction Demo"
   this.Left = 1.5
   this.ScrollBar = 2
   this.Top = 0.8818
   this.Height = 17.7051
   this.Width = 94.166

   DEFINE PUSHBUTTON PB_BEGINTRANS OF THIS;
       PROPERTY;
         Text "BeginTrans()",;
         Group .T.,;
         Left 26,;
         OnClick CLASS::PB_BEGINTRANS_ONCLICK,;
         Top 15.4102,;
         Height 1.6475,;
         Width 14.165

   DEFINE PUSHBUTTON PB_COMMIT OF THIS;
       PROPERTY;
         Enabled .F.,;
         Text "Commit()",;
         Group .T.,;
         Left 41,;
         OnClick CLASS::PB_COMMIT_ONCLICK,;
         Top 15.4102,;
         Height 1.6475,;
         Width 14.165

   DEFINE PUSHBUTTON PB_ROLLBACK OF THIS;
       PROPERTY;
         Enabled .F.,;
         Text "Rollback()",;
         Group .T.,;
         Left 56,;
         OnClick CLASS::PB_ROLLBACK_ONCLICK,;
         Top 15.4102,;
         Height 1.6475,;
         Width 14.165

   DEFINE BROWSE BROWSE1 OF THIS;
       PROPERTY;
         CUATab .T.,;
         Left 1.5,;
         Top 0.293,;
         Height 14.1172,;
         Width 89.165,;
         FontBold .T.

   Procedure PB_BEGINTRANS_OnClick
      if .not. form.intrans
         if begintrans()
            form.pb_commit.enabled = .t.
            form.pb_rollback.enabled = .t.
            this.enabled = .f.
            form.intrans = .t.
         endif
      endif

   Procedure PB_COMMIT_OnClick
      if form.intrans
         commit()
         form.intrans = .f.
         form.pb_rollback.enabled = .f.
         form.pb_begintrans.enabled = .t.
         this.enabled = .f.
      endif

   Procedure PB_ROLLBACK_OnClick
      if form.intrans
         rollback()
         form.intrans = .f.
         form.pb_commit.enabled = .f.
         form.pb_begintrans.enabled = .t.
         this.enabled = .f.
      endif

   Procedure Form_OnOpen
      form.intrans = .f.

ENDCLASS
```

Listing A, on the previous page, contains the code for a more sophisticated demonstration, TTEST.WFM. This form contains a BROWSE control for editing data from the FLIGHTS table, and one PushButton each for BeginTrans( ), Commit( ), and RollBack( ). You can run this form and easily experiment with the effectiveness of RollBack( ). Figures A, B, and C, on page 12, show a typical sequence of events from BeginTrans( ) through RollBack( ).

Making a commit( )-ment

A couple of notes are in order before we close. If you're using Visual dBASE for client/server work, you can include an optional parameter to specify the isolation level in the BeginTrans( ) function. Consult the documentation on your back-end database, as well as dBASE's Help file, for more information.

Finally, if you append records during a transaction and then issue a call to RollBack( ), dBASE marks the new records for deletion, but doesn't permanently remove them from the DBF table. This behavior might or might not be true for client/server databases. Again, consult the references you have for your database server.

You can easily employ transactions in your applications; they can add an important new ability that would be very difficult to code manually. You'll want to check out this great capability to save development time. ❖

PROGRAMMING TIP

An array of possibilities

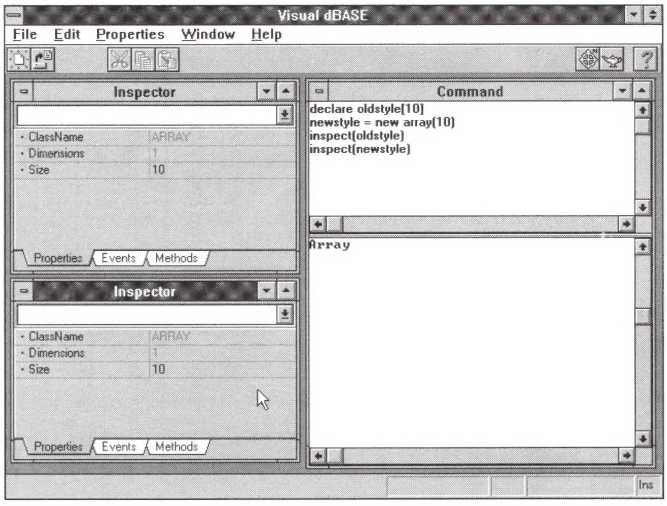
Visual dBASE extends the old DOS-style arrays in many ways, and almost all of them are significant and useful. The most important change is that arrays are now objects, complete with properties and methods. In fact, you can't make an array that isn't an object, even if you use the old DOS syntax!

Since an array is an object, you can use dBASE's property inspector to view its properties. To demonstrate, enter the following code in the Command window:

```
declare oldstyle[10]
newstyle = new array(10)
inspect(oldstyle)
inspect(newstyle)
```

You've just created two arrays, one with the DOS-style DECLARE syntax (oldstyle), and one with Visual dBASE's NEW syntax (newstyle). DBASE invokes two inspector windows, one for each array. Move the inspector windows so that both are in view, as shown in Figure A. Then, compare the two by clicking on the Properties tab of each inspector window. There aren't any differences! So whether you employ DECLARE, NEW, or DEFINE syntax, you'll get the same array object as a result.

Figure A



New-style and old-style syntax both produce array objects; use the object inspector to view an array's properties.

Array properties

While you've got those inspector windows open, let's take a look at the properties and methods packaged with an array object. An array has just three properties. The first, ClassName, will always be Array—it's read-only, so you can't change it. The Dimensions property indicates the number of dimensions or "columns" you used to create the array. This property is also read-only. The Size property tells how many elements are currently stored in the array. This property changes as the array's size changes.

You can change the Size property to change the size of an array, as long as the array contains only one

dimension. We don't recommend doing so, however, since an array contains several built-in methods that are better resizing tools.

We're usually most interested in the Size property. Since each array keeps track of its own size, we can process every element in an array very easily by using a FOR or DO WHILE loop in the form

```
*Build a Sine wave in an array
A = NEW Array(50)
FOR I = 1 TO A.Size
   A[I] = SIN(I) * 10
NEXT
```

An array of methods

There are more than a dozen methods in an array object. In one of the open inspector windows, select the Methods tab, as shown in Figure B. Now, let's discuss and play with a few of the most frequently used methods.

Add( )

The Add( ) method adds a single element to a one-dimensional array. You supply the value to add as an argument, as follows:

```
MyArray.Add("Here's a new array element!")
```

Grow( )

Grow( ) also adds to an array, but it works with one- or multi-dimensional arrays. The argument to Grow( ) can be the value 1 or 2. If it's 1, Grow( ) adds an element to a one-dimensional array. If the argument is 2, Grow( ) adds enough elements to comprise a new row in a multi-dimensional array. You can demonstrate this behavior by entering the following commands:

```
A = NEW Array(0) && Create a new array with 0 elements
A.Grow(1)  && Extends A by 1 element

B = NEW Array(1,1) && Create a two-dimensional array
B.Grow(2) && Extends B by 2 elements, or 1 row
```

Delete( )

The Delete( ) method works a little differently than you might expect from its name. Delete( ) doesn't actually resize an array, but instead rearranges it so that the "deleted" element changes to a value of

Figure B

Here are the methods associated with an array.

INSIDE VISUAL dBASE™

Inside Visual dBASE (ISSN 1084-1970) is published monthly by The Cobb Group.

Staff:
Contributing Editor-in-Chief ..... Keith G. Chuvala
Associate Editors-in-Chief ..... Jeff E. Davis
Tiffany M. Taylor
Publications Coordinator ..... Maureen Spencer
Editors ..... Laura Merrill
Joan McKim
Elisabeth Pehlke
Production Artist ..... Alison Schwarz
Product Group Manager ..... Mike Stephens
Circulation Manager ..... Mike Schroeder
Associate Publisher ..... Mark Kimbell
VP/Publisher ..... Mark Crane

Address:
Please send tips, special requests, and other correspondence to
The Editor, *Inside Visual dBASE*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: visual_dbase_win@merlin.cobb.zd.com

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to
Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: customer_relations@merlin.cobb.zd.com

Phone:
Toll free US ..... (800) 223-8720
Toll free UK ..... (0800) 961897
Local ..... (502) 493-3300
Customer Relations Fax ..... (502) 491-8050
Editorial Department Fax ..... (502) 491-3433

Back Issues:
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $8.50 each, $8.95 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Postmaster:
Second Class Postage Paid in Louisville, KY.
Postmaster: Send address changes to:
Inside Visual dBASE
P.O. Box 35160
Louisville, KY 40232

Advertising:
For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Prices:
Domestic ..... $79/yr ($8.50 each)
Outside US ..... $89/yr ($8.95 each)

.F. and moves to the end of the array. The array's actual size doesn't change. To illustrate this point, enter the following instructions in the Command window:

```
A = new array(5)
A[1] = "Value 1"
A[2] = "Value 2"
A[3] = "Value 3"
A[4] = "Value 4"
A[5] = "Value 5"
A.Delete(3)
display memory
```

When you do, dBASE will display the following output in the Results pane:

```
User Memory Variables

A               Pub   A  [10]
          [   1] C  "Value 1"
          [   2] C  "Value 2"
          [   3] C  "Value 4"
          [   4] C  "Value 5"
          [   5] L  .F.
```

Note how "Value 3" moves to the end of the array and takes the value .F. Like Grow( ), the Delete( ) method works on one- or multi-dimensional arrays. In the latter case, you can use it to delete entire rows or columns from the array.

Resize( )

You use the Resize( ) method to expand or shrink an array in just about any way. Unlike Delete( ), if you delete rows or columns with Resize( ), dBASE actually removes the elements from the array and adjusts the array's size accordingly.

Fields( )

The Fields( ) method resizes the array so that it's two-dimensional—with four columns and one row for each field in the current table—then automatically fills the elements with the structure of the table. That's a lot of work for one method! To demonstrate, enter these commands in the Command window:

```
USE \vdb\samples\animals
A = new array()
A.fields()
display memory
```

Then, dBASE displays this output in the Results pane:

```
User Memory Variables

A               Pub   A  [5,4]
      [   1,   1] C  "NAME"
      [   1,   2] C  "C"
      [   1,   3] N          10.00 (10.0000000000000)
      [   1,   4] N           0.00 (0.00000000000000)
      [   2,   1] C  "SIZE"
      [   2,   2] C  "N"
      [   2,   3] N           2.00 (2.00000000000000)
      [   2,   4] N           0.00 (0.00000000000000)
      [   3,   1] C  "WEIGHT"
      [   3,   2] C  "N"
      [   3,   3] N           2.00 (2.00000000000000)
      [   3,   4] N           0.00 (0.00000000000000)
```

DIR( )

Dir( ) is another nifty method that automatically fills an array with values. It reads the current directory information from disk into the array. Then, dBASE creates columns for filename, size, date, time, and file attributes. You can demonstrate this method by entering the commands

```
A = new array()
A.Dir()
display memory
```

When you do, you'll see the following output in the Results pane:

```
User Memory Variables

A               Pub   A  [23,5]
      [   1,   1] C  "CLIENTS.DBF"
      [   1,   2] N       18782.00 (18782.0000000000)
      [   1,   3] D  05/23/96
      [   1,   4] C  "20:12:12"
      [   1,   5] C  ".A..."
      [   2,   1] C  "PAINT.WFM"
      [   2,   2] N         551.00 (551.000000000000)
      [   2,   3] D  05/29/96
      [   2,   4] C  "17:27:14"
      [   2,   5] C  ".A..."
      [   3,   1] C  "ORDERS.DBF"
      [   3,   2] N        5634.00 (5634.00000000000)
```

If you're using Windows 95, you might want to use the DirExt( ) method, which understands and works properly with long filenames and other attributes available in Windows 95. If you support both Windows 3.x and Win95 environments, you can determine which environment your users are running with the value returned by the OS(1) function call. Next month, we'll show you some other ways you can use arrays in your applications. ❖